VSIPL++: Serial and Parallel Performance

Mark Mitchell
Jeffrey Oldham
Nathan Sidwell
CodeSourcery, LLC

May 22, 2003

1 Introduction

The VSIPL (the Vector, Signal, and Image Processing Library) specification defines a portable, C programming language interface to use in linear algebra and signal-processing applications. The VSIPL standard has been implemented by a variety of vendors. VSIPL's portable interface provides developers the ability to write code once and reuse it in multiple environments.

At HPEC 2002, we presented an overview of VSIPL++, a C++ specification designed to perform the same types of computations as VSIPL. The primary goals for VSIPL++ are improved serial performance relative to VSIPL, support for multi-processor systems, extensibility, and simpler syntax.

The serial VSIPL++ specification is virtually complete. By HPEC 2003, we expect to have a successful implementation of the specification. We anticipate that the performance of the VSIPL++ reference implementation will be superior to that of VSIPL for some applications. By HPEC 2003, the reference implementation of VSIPL++ will contain preliminary support for parallel systems.

Our presentation will compare the performance of VSIPL++ with VSIPL, and demonstrate the VSIPL++ support for parallel computation. We will also discuss VSIPL++ implementation strategies, including the use of an existing VSIPL implementation, a native C++ implementation using expression-templates, and a hybrid approach that allows an implementor to incrementally reimplement portions of VSIPL++ to achieve higher performance.

2 Performance Comparisons

VSIPL++ can be implemented on either uni-processor or multi-processor hardware. As a first step, we are implementing VSIPL++ using an existing C VSIPL library. While this implementation is straightforward, the performance is of course limited by the performance of the underlying VSIPL implementation. We also have a preliminary implementation of some portions of VSIPL++

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
20 AUG 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
VSIPL++: Serial and Parallel Performance	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **CodeSourcery, LLC**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**UU**	**16**	

using a high-performance expression-template technique. By HPEC 2003, we plan to have a partial parallel implementation of VSIPL++.

We are using a simple FIR-filter and narrowband beamforming application as a benchmark. These computations are fundamental to many signal-processing applications. We plan to present performance comparisons between VSIPL, VSIPL++ built atop VSIPL, VSIPL++ using expression templates, and VSIPL++ using multiple processors.

3 Parallel Computation Model

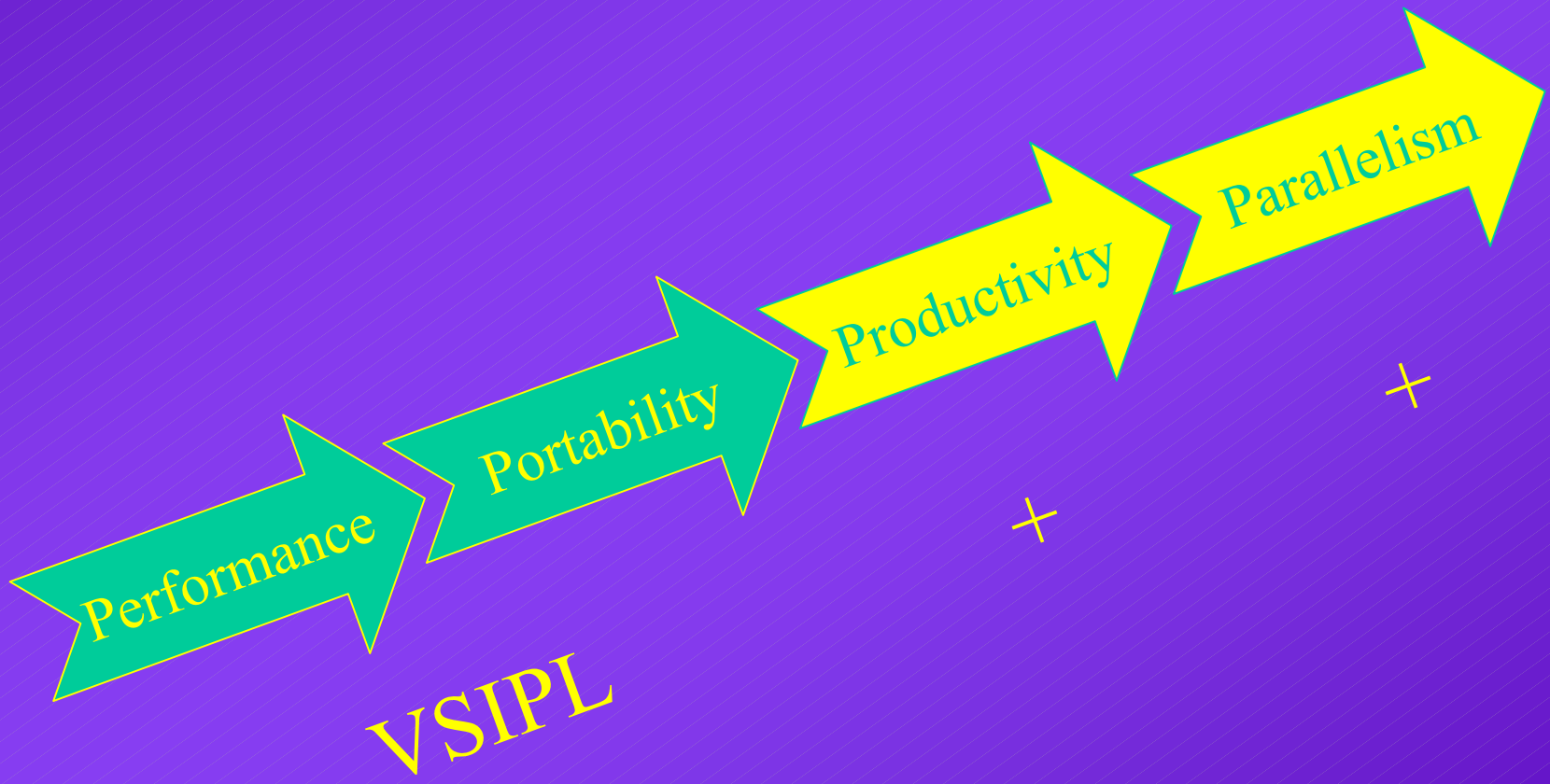
VSIPL++ uses a Single Program Multiple Data (SPMD) model when performing parallel computations. The VSIPL++ model divides rectangular arrays of data (known as "blocks") into sections using combinations of block and cyclic data distributions. We will explain the VSIPL++ model, and demonstrate how a very simple distribution model can accommodate systems ranging from small embedded systems to large systems with thousands of nodes. We will also explain how a wide variety of distribution policies can be implemented atop the simple distribution model provided by VSIPL++. Finally, we will explain how the VSIPL++ parallelism model provides support for fault-tolerance via dynamic reallocation of processors.

Serial and Parallel Performance

CodeSourcery, LLC

September 23, 2003

Design Path


Performance
Portability
Productivity
Parallelism
+
+
VSIPL

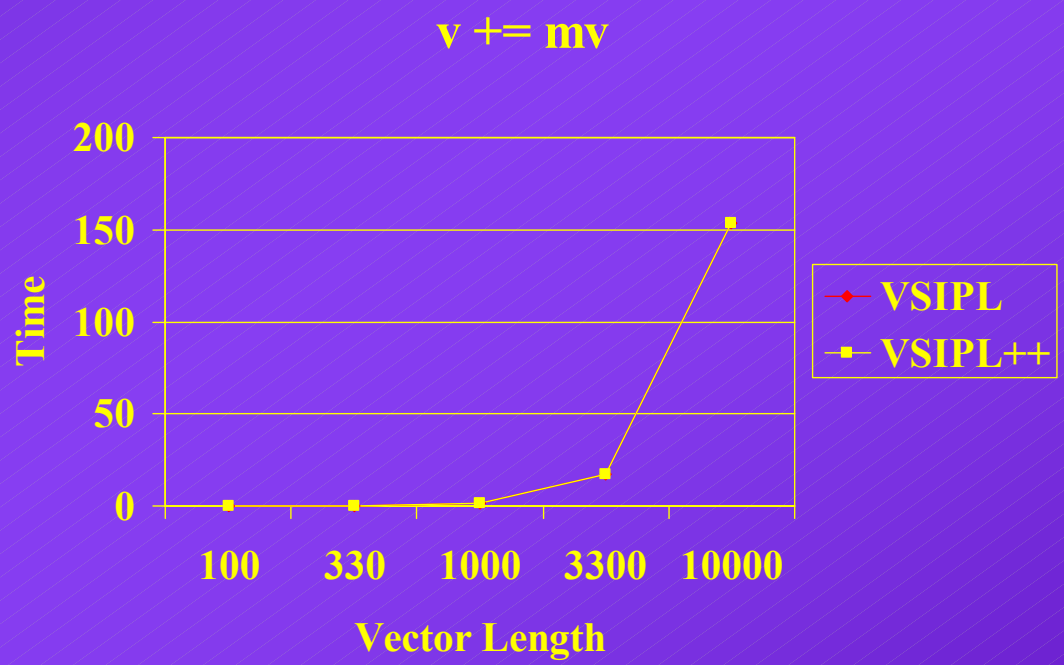
Specification Status

- Serial Specification
  - 216-page draft.
  - Under review by VSIPL Forum.
- Parallel Specification
  - 24-page preliminary draft.
  - Initial conceptual review complete.

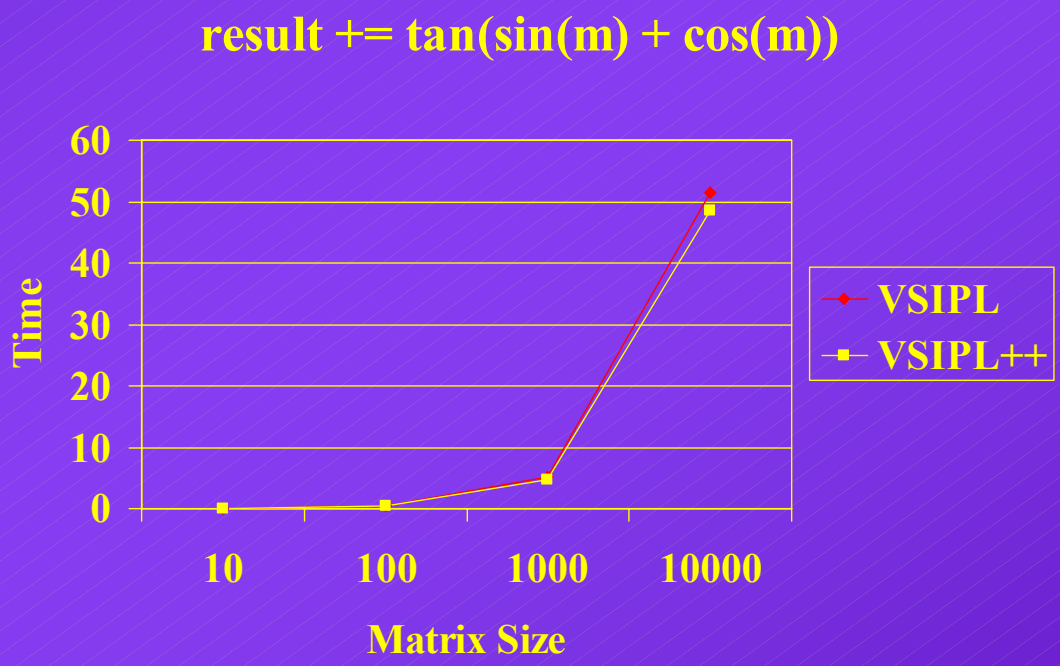
Serial Performance

- Uses VSIPL reference implementation.
  - Not the fastest implementation…
  - … but the relative performance is important.
- Environment:
  - 2GHz Pentium-M
  - 512KB cache, 512MB RAM
  - GNU/Linux, G++ 3.4

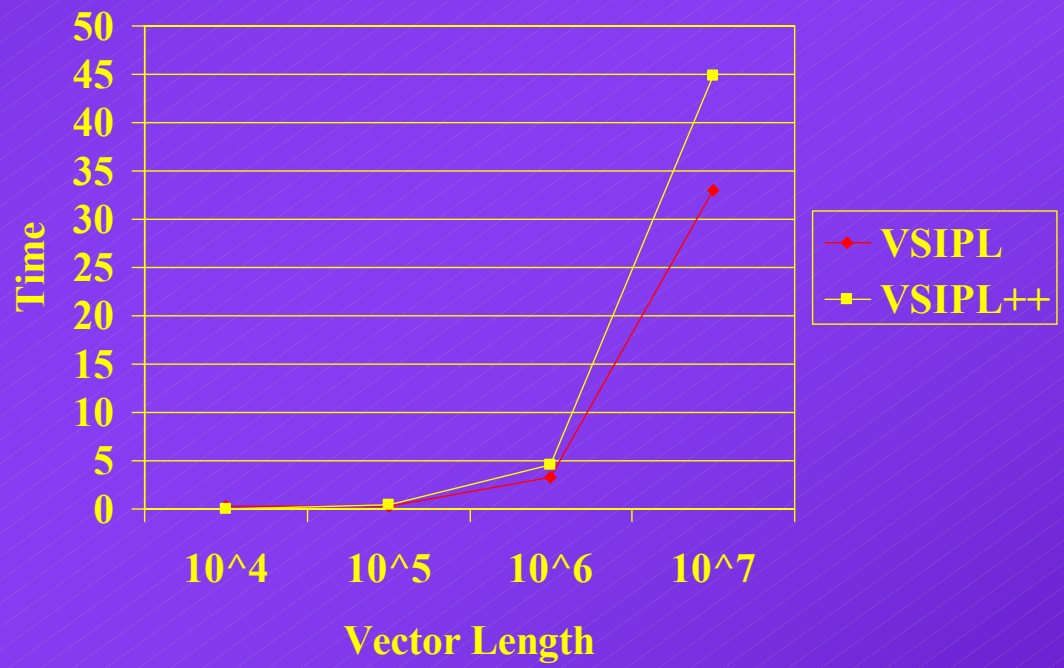
Matrix/Vector



Matrix/Matrix



Checked Vector Access



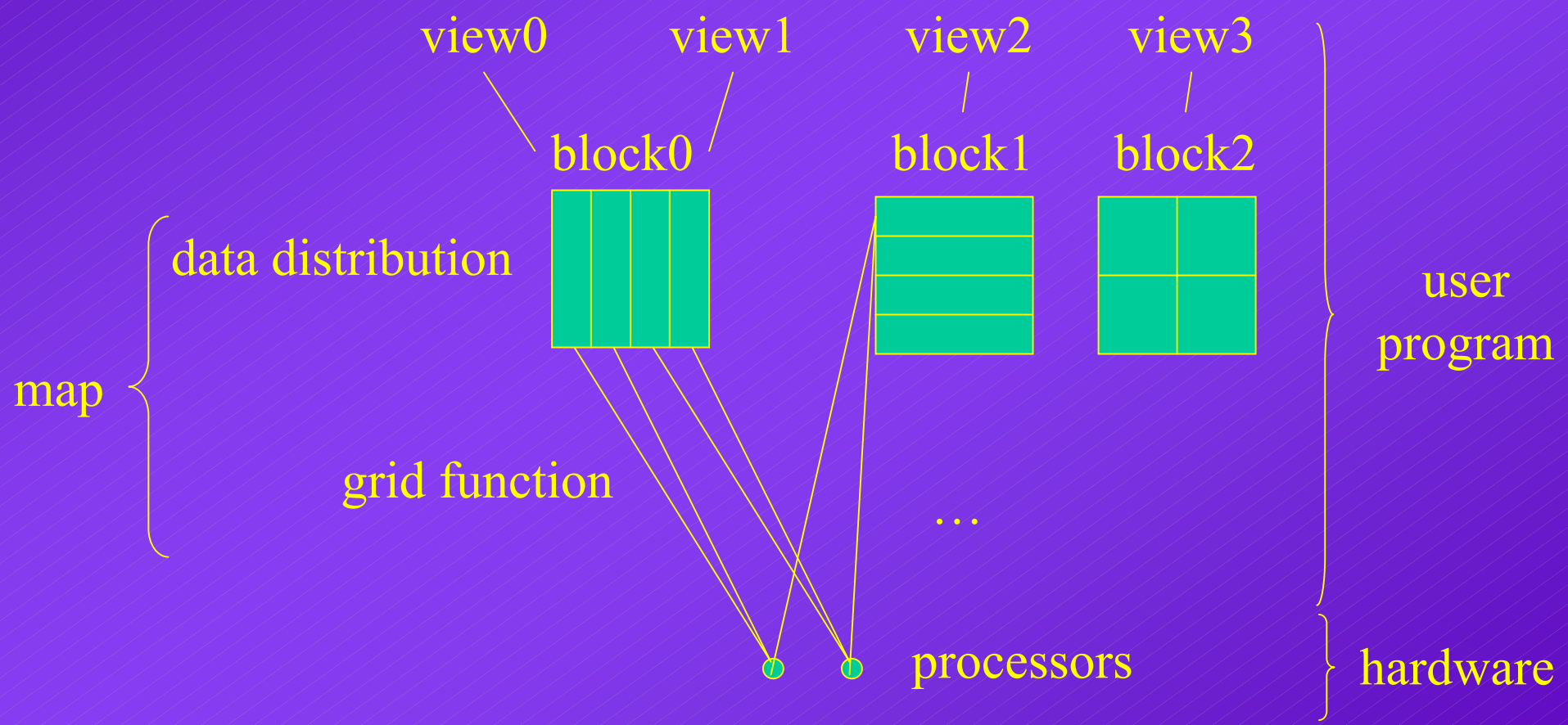
Performance Conclusions

- VSIPL++ has approximately zero overhead.
  - Memory effects actually enable VSIPL++ to outperform VSIPL.
  - Expression-template techniques may also improve performance.
- Exceptions are expensive.
  - We are not sure if this overhead can be eliminated.
- Reference implementation will be directly useful.
  - Vendor-optimized versions will probably be better.

Parallelism

- Target systems:
  - Support 1-64K+ processors.
  - Support MPI, POSIX threads.
- Conceptual model:
  - Single-program multiple-data model.
  - Owner computes.
  - Parallelism requires changing only declarations, not expressions.

Parallel VSIPL++ Model



Using Parallelism

- Declaration:

```
Vector<double,
        Dense<1, double,
                Map<Block> > >
v (17, 1.0, Block(4));
```

- Meaning:
  - 17: Vector length.
  - 1.0: Intial value.
  - `Block(4)`: Block distribution over 4 processors.

FYO4 Objectives

- Specification:
  - Finalize serial and parallel specifications.
  - Get approval from VSIPL Forum.
- Implementation:
  - Finish serial implementation.
  - Draft parallel implementation.
- Measurement:
  - Performance analysis.

Contact Information

- Mark Mitchell
  mark@codesourcery.com
- Jeffrey Oldham
  oldham@codesourcery.com
- Nathan Sidwell
  nathan@codesourcery.com

Serial and Parallel Performance

CodeSourcery, LLC

September 23, 2003